Automatic Grid Service Generator

(Version 1.0)

USER MANUAL

Table of Contents

Table of Contents....3
1. Introduction....4
1.1 Grid Services Overview....4
1.2 Building a Grid Service....4
2. AGSG Tool....6
2.1 Availability and Pre Requisites....6
2.2 Accessing the AGSG Tool....6
2.3 AGSG Features....6
3. Generating Grid Service & Corresponding Client....8
3.1 Service Creator....8
3.2 Service Deployment....15
3.3 Service Invoker....16
4. Contact Details....18

1. Introduction

The Automatic Grid Service Generator (AGSG) is a web tool to automatically generate Grid services. It can convert an existing executable file as a grid service. The AGSG tool can work on any Globus 4.x based grid.
This tool is useful for application developers or programmers who wish to build a Grid Service from his/her application.

1.1 Grid Services Overview

Grid Services are like Web Services but with improved characteristics to address fundamental issues in distributed (grid) computing relating to how to name, create, discover, monitor, and manage the lifetime of stateful services. Grid Services are Stateful and potentially transient services, support Notifications, PortType extension, Lifecycle management etc.A Grid Service adheres to the WSRF/OGSI specification. The layered diagram below shows the relationship between the Open Grid Services Architecture (OGSA), Web Services Resource Framework (WSRF), and Globus Toolkit 4 (GT4).

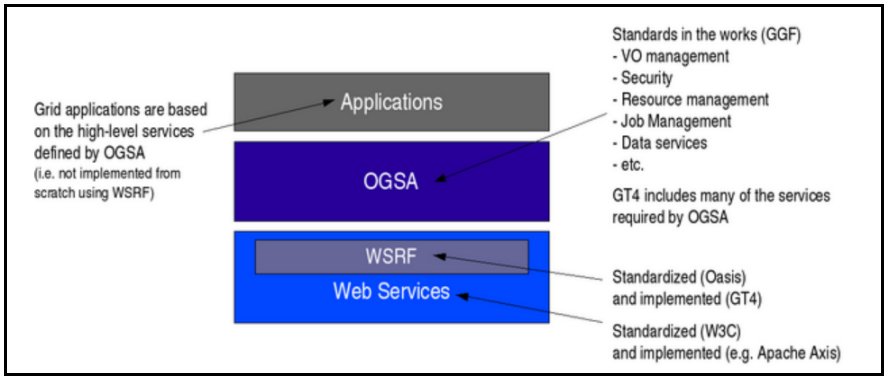

Grid applications are based on the high-level services defined by OGSA (i.e. not implemented from scratch using WSRF)
Applications
OGSA
WSRF
Web Services
Standards in the works (GGF)
- VO management
- Security
- Resource management
- Job Management
- Data services
- etc.
GT4 includes many of the services required by OGSA
Standardized (Oasis) and implemented (GT4)
Standardized (W3C) and implemented (e.g. Apache Axis)

Figure 1: Layered Relationship Diagram

1.2 Building a Grid Service

Following are the basic steps to create a Grid Service in GT4.

1. **Define the service's interface**.
Define the public interfaces for the service. This is done with *GWSDL*

2. **Implement the service**.
Write the server & client source files describing the implementtaion of the interface. This is done with *Java.*

3. **Define the deployment parameters**.
Specify the container-level service configuration information. This is done with *WSDD and JNDI*

4. **Compile everything and generate GAR file**.
This is done with *Ant.* Ant is an open source tool similar to “make” that helps to build large projects.

5. **Deploy service**.
The service will execute in a hosting environment (i.e. the service container of the Globus Toolkit 4). Hence, the .gar file has to be deployed inside the globus container.

Once the grid service is deployed, a URI (Uniform Resource Identifier) is generated and multiple clients/users can invoke the service using the URI.

The URI will be like this:

`https://<IP_Address:Port_Number>/wsrf/services/<ApplicationPackageName>/<ServiceName>Service`

For Example:-
If “Math” is service name and package of application is “example/core/first” then URI will be
`https://192.168.60.168:8443/wsrf/services/examples/core/first/MathService`

2. AGSG Tool

2.1 Pre Requisites

User should preferably use Browser IE Version 6.0 and above or Mozilla Firefox 3.0.7 and above.

2.2 Accessing the AGSG Tool

The tool can be accessed from any machine connected to the GARUDA grid through the Garuda Portal.

Note: *-*
Garuda Portal URI is http://192.168.60.40/GridPortal1.3/ *on Garuda network or* http://203.200.36.236/GridPortal1.3 *via internet*

In order to access the AGSG tool, click the **"Service Generator"** link in the Garuda Portal.
The **"Service Generator"** link has two sub links called **"Service Creator"** and **"Service Invoker"**

- Click **"Service Creator"** link for creating a new Grid Service and
- Click **"Service Invoker"** link for invoking a published service using its URI.

2.3 AGSG Features

The Automatic Grid Service Generator tool can generate Grid services from already existing executable files.

It frees the user from the complexity involved in making a grid service (steps discussed in Section 1.2).

The AGSG tool internally creates files such as GWSDL, WSDD, JNDI, and Service Implementation, which are required while generating a grid service.

The features of AGSG are:-

- Web Based tool.
- User Friendly GUI.
- Frees user from the complexity involved in making a grid service
- Converts an executable file as a grid service.
- Support executables of different languages (C, C++, Fortran 77&90).
- Supports both compiled and interpreted language applications.
- Globus 4.x based grid.

- Client stub generation.
- Client code generation (Required for invoking a service).
- Generic Interface tested with different types of applications having different types and number of I-O parameters
- Enables Service generation and deployment on heterogeneous resources of GARUDA

3. Generating Grid Service & Corresponding Client

3.1 Service Creator

The **"Service Creator"** helps to automatically build a Grid Service from an executable file.

The GUI for Service Creator is shown below.

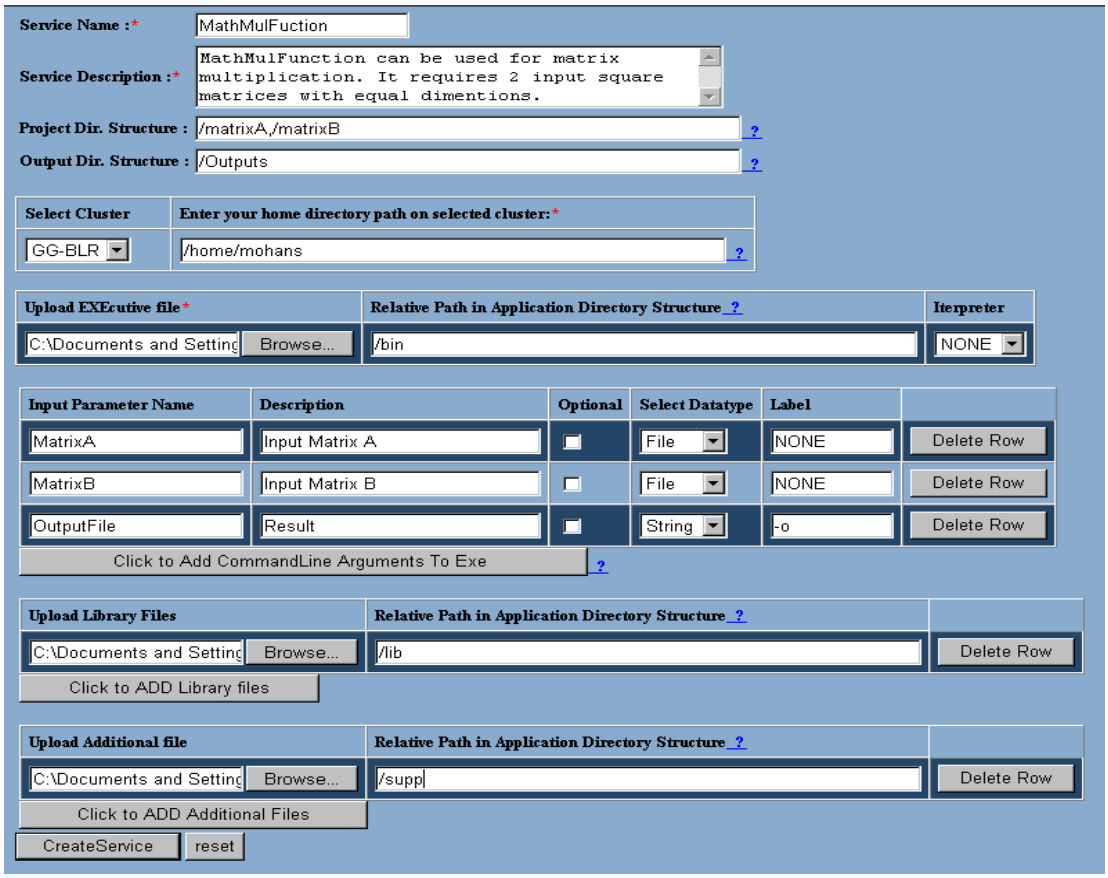


Figure 2: User Interface

- In order to create a service, the user needs to provide the following information
  - Service Name
  - Service Description
  - Select cluster and Enter home directory on selected cluster.
  - Upload Executable file
  - Specify Interpreter
  - Input Parameters
  - Project Directory Structure (optional)
  - Output Directory structure (optional)
  - Supported Files (Optional)

- Mandatory fields are marked with *.
- Online help can be obtained for each feature by clinking the '?'

Each of these fields are illustrated with an example in this document

3.1.1 Basic Service Creation

i) Service name
The user should give a name for the service to be generated. It should be unique. The service generated will be identified with this name.

For Example, if service name is given as 'MathMulFunction'.

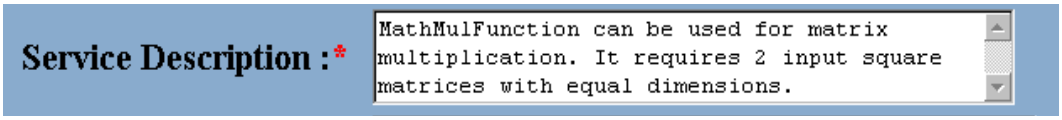
The URI (Uniform Resource Identifier) generated for the service will be like this:

https://<IP_Address:Port_Number>/wsrf/services/<ApplicationPackageName>/<ServiceName>Service

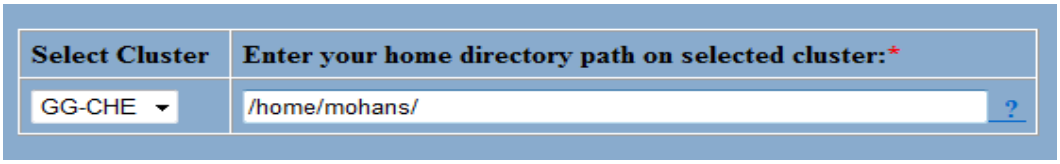
For Example:-
If "MathMulFunction" is service name and package of application is "example/core/first" then URI will be
https://192.168.60.168:8443/wsrf/services/examples/core/first/MathMulFunctionService

ii) Service Description
In this field the user should give a brief description of what the service offers, so that people interested to use this service can understand what it does.

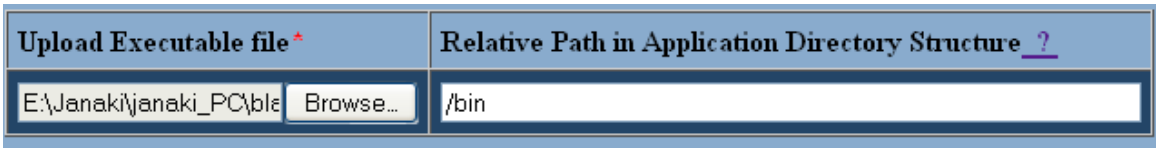
Ex: MathMulFunction is used for matrix multiplication


Service Description :*
MathMulFunction can be used for matrix multiplication. It requires 2 input square matrices with equal dimensions.

iii) Select Cluster and Enter Home Directory
In this field the user should select the cluster, where the application is pre compiled, and he wants to use this pre compiled executable of his application for creating a Grid Service.
User also needs to provide his home directory path in the selected cluster.

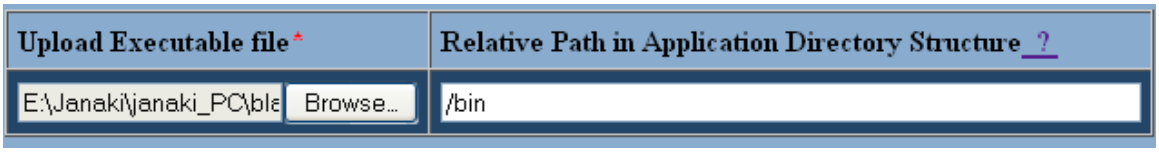


iv) Upload Executable file
User needs to upload the required executable file.



If user is using an interpreted language to develop his application, then the corresponding interpreter must be specified in the provided field.

iv) Relative Path in Application Directory Structure (Optional)
Some applications require that the executable should reside in particular directory, In such cases, the path can be supplied with this field "**Relative Path in Application Directory Structure**".



v) Input Parameters
The input parameters (if any) required by the application, can be given in this field. There may be some applications that do not require any input and some applications which require multiple inputs.
For Example:

a) Executable name followed by either one input or multiple inputs
   Ex: **`./exec inputfile1 inputfile2`**
   Input can be file or any datatype supported by java language.
b) Executable followed by multiple parameters.
   Ex: **`./exec –i inputfile1 –p1 param1 –p2 param2 –p3 param3 -p4 param4`**
   Input parameters having options as pvalue1 having option –param1
c) Executable followed by input and multiple parameters
   Ex: **`./exec inputfile1 –p1 param1 –p2 param2 –p3 param3`**
d) Executable with input as stdin
   Ex: **`./exec < inputfile1`**
e) Executable followed by parameter values

Ex: **`./exec param1 param2 param3`**

f) Redirecting default output (stdout) to a file.
.Ex: **`./exec > outfile`**

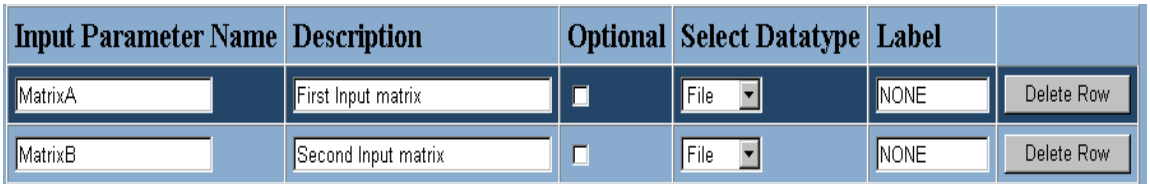
g) Executable only. Input file taken by default. Input file should be available in current working directory.

Case a)

`./exec inputfile1 inputfile2`

Where the `inputfile1` and `inputfile2` are parameters of the type file.

Suppose an executable accepts two matrices i.e. MatixA and MatrixB as input file for addition

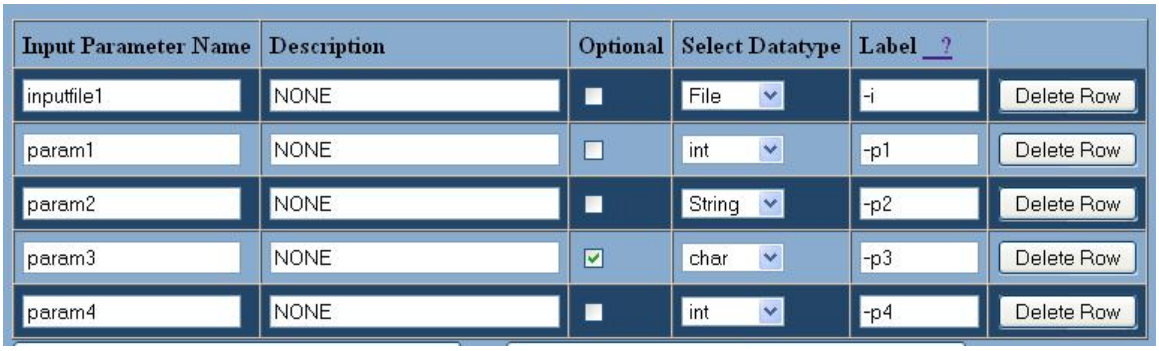


Note: Input parameter name can be given according to user's choice. Datatype 'File' can only be given in case user uploads any file as input from client code.
If input file is already present in the current working directory (when file is uploaded as supported file) then the Datatype of input file should be 'String'.

Case b)

`./exec –i inputfile1 –p1 param1 –p2 param2 –p3 param3 -p4 param4`

In case b), the application executable needs multiple input parameters. In such case user needs to give the types of each parameter value of each parameter and order (if any) to be maintained. Then the parameters need to be specified in AGSG interface. Suppose if parameter for the application is optional, i.e. the client may/may not use this parameter while invoking the service. Such optional parameters of the service can be specified by ticking the 'optional' check box
Description for each parameter can be provided in *Description* field.



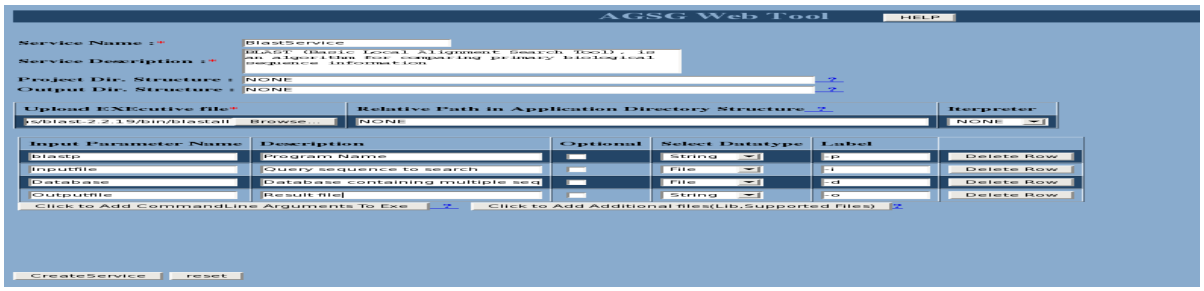
For example:
The Basic Local Alignment Search Tool (BLAST) is a bio informatics application which does sequence analysis on protein database to search for a sequence of Amino Acid.

The command line execution for BLAST application is
$./executable -p <program name> -i <File Containing query string> -d <Database file> -o <Output file>

```
./blastall -p blastp -i hahu.aa -d yeast.aa -o outfile
```

-p is for program name.
-i input file name - contains query sequence to search.
-d Database file - sequence database file containing multiple sequences.
-o output file name.

So the inputs parameters required for this application is specified in the AGSG interface as shown below.



Case c)

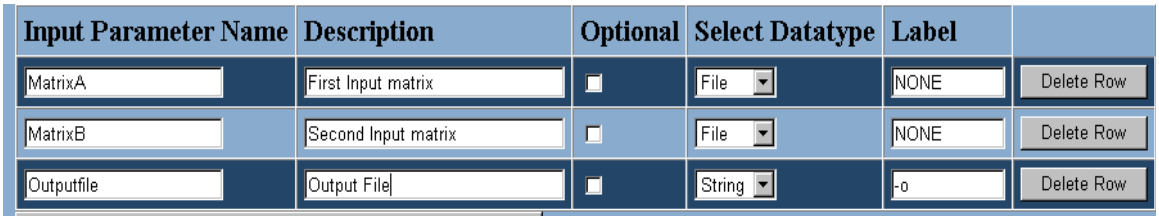
```
./exec inputfile1 –p1 param1 –p2 param2 –p3 param3
```

In some cases the input parameter to executable have label. As param1,param2,pram3 have labels –p1, -p2 and –p3 respectively. Inputfile1 does not have label.

Suppose an executable accepts two matrices i.e. MatixA and MatrixB as input file for addition. And result will be produce in output file, which have label '-o'. i.e.

```
./exec MatrixA MatrixB –o Outputfile
```

Then input parameters required for this application is specified in the AGSG interface as shown below.



Case d)

```
./exec < inputfile1
```

Executable with input as stdin

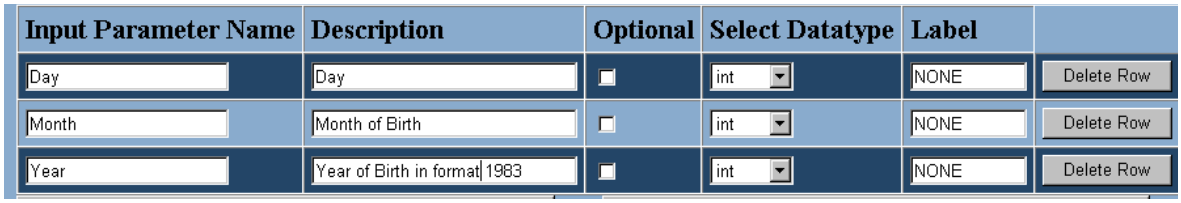
Suppose an executable accepts a matrix i.e. MatixA as input file from stdin to calculate Transpose of the matrix.
i.e. `./exec < MatrixA`
Then inputs parameters required for this executable specified in the AGSG interface as shown below.

Case e)

`./exec param1 param2 param3`

Executable followed by parameter values.

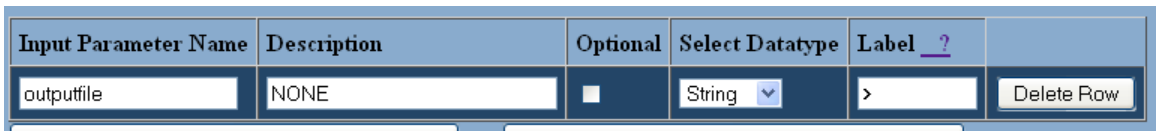
Suppose an executable calculate age of the user by accepting date of birth as input.
i.e. `./exec Day Month Year`
Then inputs parameters required for this executable specified in the AGSG interface as shown below.



In this example, there are no **`labels`** given to any of the parameters; only the **`Description`** and **`Datatype`** of each parameter is specified.

Case f)

In case, the output from the default output (**`stdout`**) needs to be redirected to a particular output file.



Case g)

In case, no parameter is required for the executable to run and by default if it picks the required input file from current directory, there is no need to give any Command Line arguments.

3.1.2 Advanced Service Creation

When some application runs, it requires a specific directory structure to be present at run time. This specific directory structure can be given as explained below.

vi) Project Directory Structure - Optional

If user wants the executable, input files and any supported library files under one project directory, the structure of project directory can be specified by him. For example, if a project directory structure contain a `/lib` and `/input`, subdirectories then it can be specified as shown in figure below. This will create the directories **`/<applicationname>/`**`lib` and **`/<applicationname>/`**`input` under the application directory.

vii) Output Directory structure (Optional)

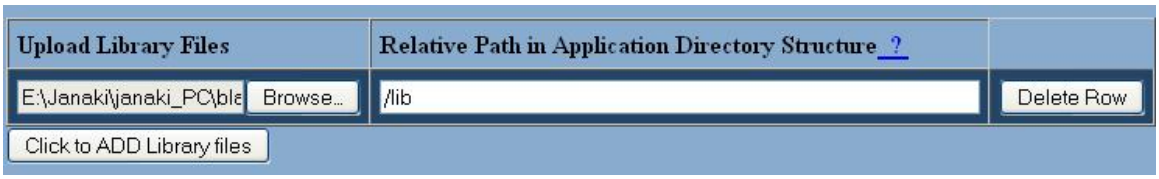
If user wants all the outputs to be placed in a particular directory, then he can mention the output directory structure.

Ex: `/Outputs`

All outputs will be sent to **`/<applicationname>/`**`Outputs`

viii) Supported Files (Optional)
a) Additional Files
If the application needs any additional or supported files, these can upload by browsing and uploading the files from the local machine.

b) Library Files
User can also upload any required library files. These library files will be linked with `LD_LIBRARY_PATH`.



User can upload multiple libraries by clicking "**`Click to add Library files`**". And if any directory structure needs to be maintained for the library files, the user can specify

the same in "**Relative Path in Application Directory Structure**". This is optional.

3.2 Service Deployment

After filling in the AGSG form, User can click on "**Create Service**" for generation of service.

This process will generate a Grid ARchive (GAR) file. The service that is generated is bundled into a GAR file. This gar file needs to be deployed into globus container on the suitable SOA based grid nodes. Since the GAR file needs to be deployed in the Globus container, it can be done only by the Grid Administrator. Hence, user needs to send an e-mail request to the GARUDA System Administrator/s of the grid center where he desires to deploy the service.

The tool automatically generates a mail along with the `gar` file attachment. Figure below shows the template for requesting deployment of service. The User only needs to provide his/her mail id in the '**From**' address field. The `gar` file sent will be deployed by the System Administrator. Once the service is deployed, the System Administrator has to send a mail notification to the user providing the 'URI' of the service. Client can use this service URI later for future job submissions.

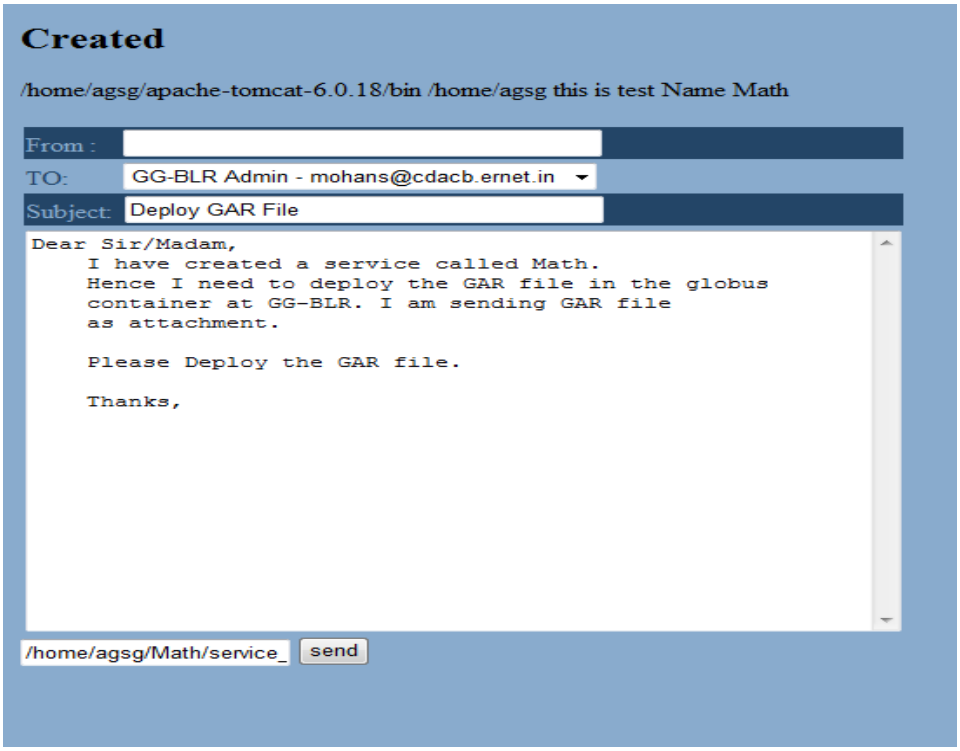


Figure 3: Request for deployment of GAR

3.3 Service Invoker

This is the *client code* to invoke the service generated; this tool provides a GUI for invoking the service.

The client can be accessed from any machine connected to the GARUDA grid through the Garuda Portal using the links
`Service Generator` → `Service Invoker`

The **"`Service Invoker`"** has a simple interface which accepts the URI of the published service and the corresponding parameters. It invokes the service.

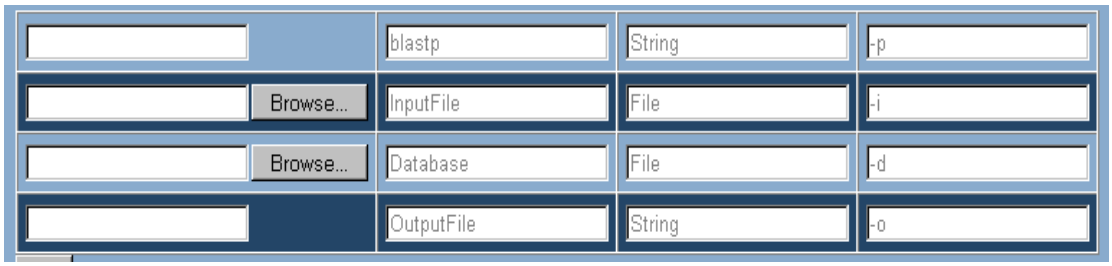
User should supply the URI in the "**`Enter Service URI:`**" Once the URI is entered and submitted, an interface will pop up showing the details of the required input parameters. The service will run on target machine and output the results.

In the following example, user at the time of service creation has given four input parameters as follows

a) `programname`, of String Datatype, with Label as -p
b) `inputfile`, of File Datatype, with Label as -i
c) `databasename`, of String Datatype, with Label as -d
d) `outputfilename`, of String Datatype, with Label as -o

Now the user (client) needs to provide the value for `programname`, `inputfile`, `databasename` and `outputfilename`.

Ex: For BLAST application, `programname` will be **`blastp`** or **`blastn`**, `databasename` will be the protein database present in the current working directory, and `outputfilename` can be of user's choice. The input file (the protein query sequence) for blast program needs to be uploaded by '**`Browse`**'.



Once the user clicks on '**`Run`**' button, the client code will run on the SOA grid node where the service is hosted. Client code will automatically invoke the Service and will pass the above values as input to the Service. Once the outputs are generated, user can download them from the following page.

Figure 4: User Interface to download output

This is the download page. From this page the user can select the output file to save the output of Service execution.

4. Contact Details

Center for Development of Advanced Computing
Address:-

'CDAC Knowledge Park', No. 1,
Old Madras Road,
Opp. HAL Aero engine Div.,
Byappanahalli, Bangalore-560038

URL:-

www.cdac.in

Email:-

ssg@cdac.in

Tel No.:-

080-66116320

Fax No.:-

080-25247724